Dialogflow ES basics

This page describes the basics of using Dialogflow ES. You should read this page before proceeding to any other documents or quickstarts.

Help users interact with technology

Traditional computer interfaces require structured and predictable input to function properly, which makes the use of these interfaces unnatural and sometimes difficult. If end-users can't easily understand this structured input, they have a hard time figuring out what to do. Ideally, your interfaces can infer what your end-users want, based on the natural language they are using.

For example, consider a simple user request like "What's the weather forecast today?". Other end-users might also ask:

- "What's the weather like right now?"
- "What's the temperature going to be in San Francisco tomorrow?"
- "What will the weather be on the 21st?"

Even with these simple questions, you can see that conversational experiences are hard to implement. Interpreting and processing natural language requires a very robust language parser. Dialogflow handles this for you, so you can provide a high quality conversational end-user experience.

Agents

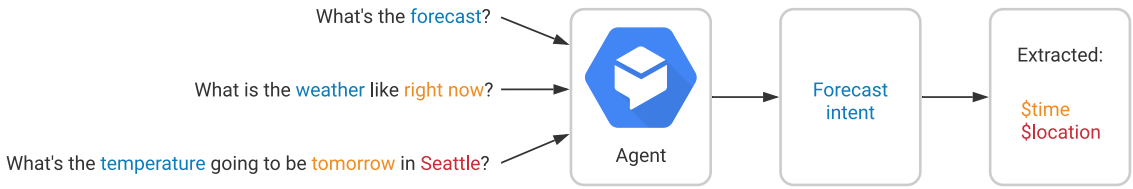
A Dialogflow agent (/dialogflow/docs/agents-overview) is a virtual agent that handles concurrent conversations with your end-users. It is a natural language understanding module that understands the nuances of human language. Dialogflow translates end-user text or audio during a conversation to structured data that your apps and services can understand. You design and build a Dialogflow agent to handle the types of conversations required for your system.

A Dialogflow agent is similar to a human call center agent. You train them both to handle expected conversation scenarios, and your training does not need to be overly explicit.

Intents

An intent (/dialogflow/docs/intents-overview) categorizes an end-user's intention for one conversation turn. For each agent, you define many intents, where your combined intents can handle a complete conversation. When an end-user writes or says something, referred to as an *end-user expression*, Dialogflow matches the end-user expression to the best intent in your agent. Matching an intent is also known as *intent classification*.

For example, you could create a weather agent that recognizes and responds to end-user questions about the weather. You would likely define an intent for questions about the weather forecast. If an end-user says "What's the forecast?", Dialogflow would match that end-user expression to the forecast intent. You can also define your intent to extract useful information from the end-user expression, like a time or location for the desired weather forecast. This extracted data is important for your system to perform a weather query for the end-user.

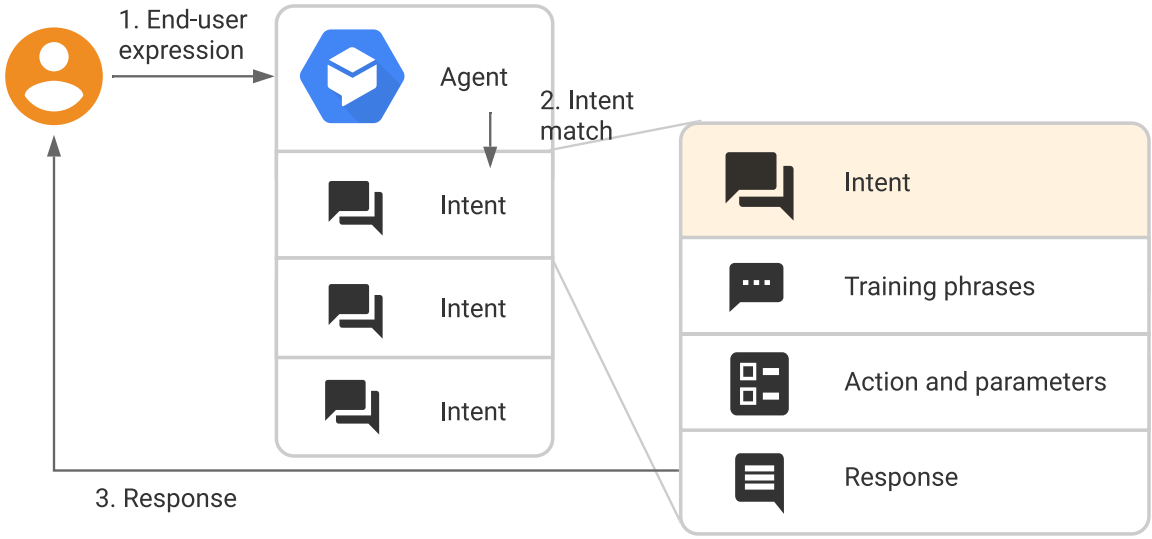


A basic intent contains the following:

- **Training phrases** (/dialogflow/docs/intents-training-phrases): These are example phrases for what end-users might say. When an end-user expression resembles one of these phrases, Dialogflow matches the intent. You don't have to define every possible example, because Dialogflow's built-in machine learning expands on your list with other, similar phrases.
- **Action** (/dialogflow/docs/intents-actions-parameters#actions): You can define an action for each intent. When an intent is matched, Dialogflow provides the action to your system, and you can use the action to trigger certain actions defined in your system.
- **Parameters** (/dialogflow/docs/intents-actions-parameters#params): When an intent is matched at runtime, Dialogflow provides the extracted values from the end-user expression as *parameters*. Each parameter has a type, called the entity type (/dialogflow/docs/entities-overview), which dictates exactly how the data is extracted. Unlike raw end-user input, parameters are structured data that can easily be used to perform some logic or generate responses.
- **Responses** (/dialogflow/docs/intents-responses): You define text, speech, or visual responses to return to the end-user. These may provide the end-user with answers,

ask the end-user for more information, or terminate the conversation.

The following diagram shows the basic flow for intent matching and responding to the end-user:



Entities

Each intent parameter (/dialogflow/docs/intents-actions-parameters#params) has a type, called the entity type (/dialogflow/docs/entities-overview), which dictates exactly how data from an end-user expression is extracted.

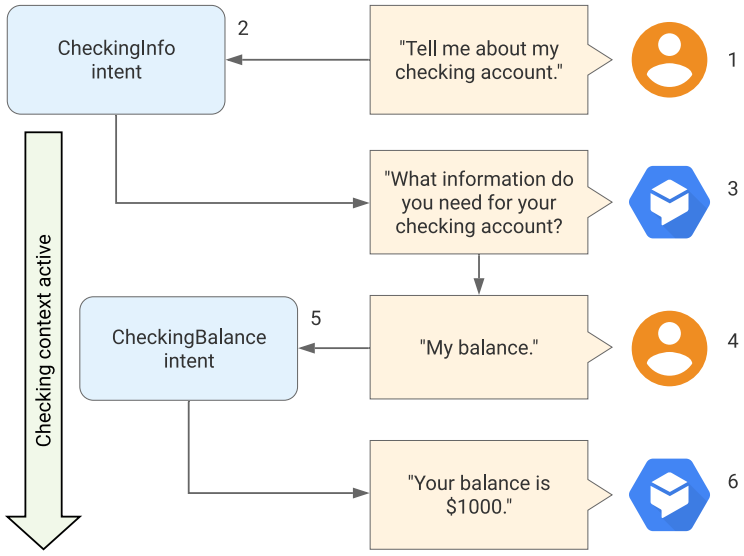
Dialogflow provides predefined system entities (/dialogflow/docs/entities-system) that can match many common types of data. For example, there are system entities for matching dates, times, colors, email addresses, and so on. You can also create your own custom entities (/dialogflow/docs/entities-custom) for matching custom data. For example, you could define a *vegetable* entity that can match the types of vegetables available for purchase with a grocery store agent.

Contexts

Dialogflow contexts (/dialogflow/docs/contexts-overview) are similar to natural language context. If a person says to you "they are orange", you need context in order to understand what "they" is referring to. Similarly, for Dialogflow to handle an end-user expression like that, it needs to be provided with context in order to correctly match an intent.

Using contexts, you can control the flow of a conversation. You can configure contexts for an intent by setting input and output contexts (/dialogflow/docs/contexts-input-output), which are identified by string names. When an intent is matched, any configured *output contexts* for that intent become active. While any contexts are active, Dialogflow is more likely to match intents that are configured with *input contexts* that correspond to the currently active contexts.

The following diagram shows an example that uses context for a banking agent.



1. The end-user asks for information about their checking account.
2. Dialogflow matches this end-user expression to the **`CheckingInfo`** intent. This intent has a **`checking`** output context, so that context becomes active.
3. The agent asks the end-user for the type of information they want about their checking account.
4. The end-user responds with "my balance".
5. Dialogflow matches this end-user expression to the **`CheckingBalance`** intent. This intent has a **`checking`** input context, which needs to be active to match this intent. A similar **`SavingsBalance`** intent may also exist for matching the same end-user expression when a **`savings`** context is active.
6. After your system performs the necessary database queries, the agent responds with the checking account balance.

Follow-up intents

You can use follow-up intents (/dialogflow/docs/contexts-follow-up-intents) to automatically set contexts for pairs of intents. A follow-up intent is a child of its associated *parent intent*. When you create a follow-up intent, an output context is automatically added to the parent intent and an input context of the same name is added to the follow-up intent. A follow-up intent is only matched when the parent intent is matched in the previous conversational turn. You can also create multiple levels of nested follow-up intents.

Dialogflow provides many predefined follow-up intents (/dialogflow/docs/reference/follow-up-intent-expressions) for common end-user replies like "yes", "no", or "cancel". You can also create your own follow-up intents to handle custom replies.

Dialogflow Console

Dialogflow provides a web user interface called the *Dialogflow Console* (visit documentation (/dialogflow/docs/console), open console (https://dialogflow.cloud.google.com)). You use this console to create, build, and test agents.

The Dialogflow Console is different from the Google Cloud Platform (GCP) Console (visit documentation (https://support.google.com/cloud/answer/3465889?hl=en&ref_topic=3340599), open console (https://console.cloud.google.com/)). The Dialogflow Console is used to manage Dialogflow agents, while the GCP Console is used to manage GCP-specific Dialogflow settings (for example, billing) and other GCP resources.

In most cases you should use the Dialogflow Console to build agents, but you can also use the Dialogflow API to build agents for advanced scenarios.

User interactions with integrations

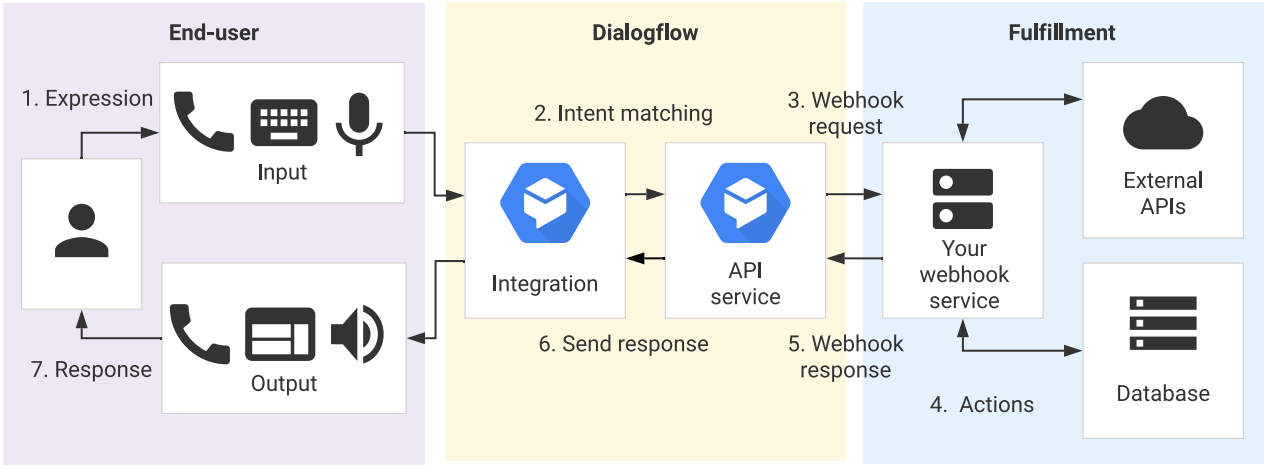
Dialogflow integrates with many popular conversation platforms like Google Assistant, Slack, and Facebook Messenger. If you want to build an agent for one of these platforms, you should use one of the many integrations (/dialogflow/docs/integrations) options. Direct end-user interactions are handled for you, so you can focus on building your agent. Each integration handles end-user interactions in a platform-specific way, so see the documentation for your integration platform for details.

Fulfillment for integrations

By default, your agent responds to a matched intent with a static response. If you're using one of the integration (/dialogflow/docs/integrations) options, you can provide a more dynamic response by using fulfillment (/dialogflow/docs/fulfillment-overview). When you enable fulfillment for an intent, Dialogflow responds to that intent by calling a service that you define. For example, if an end-user wants to schedule a haircut on Friday, your service can check your database and respond to the end-user with availability information for Friday.

Each intent (/dialogflow/docs/intents-overview) has a setting to enable fulfillment. If an intent requires some action by your system or a dynamic response, you should enable fulfillment for the intent. If an intent without fulfillment enabled is matched, Dialogflow uses the static response you defined for the intent.

When an intent with fulfillment enabled is matched, Dialogflow sends a request to your *webhook* service with information about the matched intent. Your system can perform any required actions and respond to Dialogflow with information for how to proceed. When fulfillment is enabled, the static response you defined for the intent is only used if your webhook service fails (/dialogflow/es/docs/fulfillment-webhook#errors). The following diagram shows the processing flow for fulfillment.



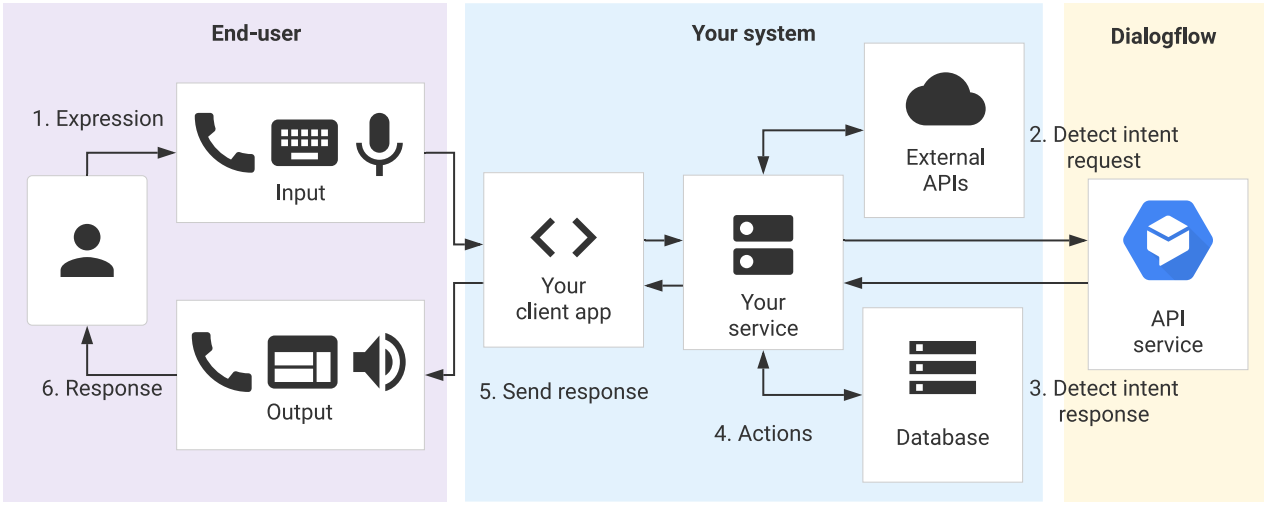
1. The end-user types or speaks an expression.
2. Dialogflow matches the end-user expression to an intent and extracts parameters.
3. Dialogflow sends a webhook request (/dialogflow/docs/fulfillment-webhook#webhook_request) message to your webhook service. This message contains information about the matched intent, the action, the parameters, and the response defined for the intent.
4. Your service performs actions as needed, like database queries or external API calls.
5. Your service sends a webhook response (/dialogflow/docs/fulfillment-webhook#webhook_response) message to Dialogflow. This

message contains the response that should be sent to the end-user.

6. Dialogflow sends the response to the end-user.
7. The end-user sees or hears the response.

User interactions with the API

If you are not using one of the integration (/dialogflow/docs/integrations) options, you must write code that directly interacts with the end-user. You must also directly interact with Dialogflow's API (/dialogflow/docs/api-overview) for each conversational turn to send end-user expressions and receive intent matches. The following diagram shows the processing flow when interacting with the API.



1. The end-user types or speaks an expression.
2. Your service sends this end-user expression to Dialogflow in a detect intent request message.
3. Dialogflow sends a detect intent response message to your service. This message contains information about the matched intent, the action, the parameters, and the response defined for the intent.
4. Your service performs actions as needed, like database queries or external API calls.
5. Your service sends a response to the end-user.
6. The end-user sees or hears the response.

Previous
← Documentation (/dialogflow/es/docs)

Next
Introduction videos (/dialogflow/es/docs/video)

Last updated 2025-01-09 UTC.